Inter-Office Memorandum

To	Dorado project	Date	April 21, 1980
From	Ed Taft	Location	PARC/CSL
Subject	Dorado booting—operation and mechanisms	File	[Ivy]<DoradoDocs>DoradoBooting.press

XEROX

Dorado bootstrapping is a complex process, involving a number of hardware, microcode, and software components. This memo attempts to document the booting mechanism thoroughly: how to use it, how it works, and how the components are constructed and maintained.

Most Dorado users will be interested only in section 1. The other sections describe arcane mechanisms and procedures that are of concern only to Dorado microprogrammers and maintainers.

1. Operation

Booting a Dorado from scratch involves two main activities: loading its control store with the correct *microcode*, and then forcing it to load and run the correct *software*. At present, the second step uses exactly the same mechanisms as are used on the Alto for bootstrapping software from the disk or Ethernet. The descriptions here concentrate on the first step.

Emulators

The Dorado is able to execute programs written in a particular language (Mesa, Lisp, Smalltalk, or Alto BCPL) by virtue of running *emulator microcode*—that is, microcode for emulating that language. In addition, there is some common *I/O microcode* for controlling the I/O devices (display, disk, Ethernet, etc.) This microcode is not built into the Dorado but rather must be loaded into its control store from some external source.

The Dorado's control store is not large enough to hold the emulator microcode for all languages simultaneously. Therefore, the microcode is divided up into several independent microprograms, only one of which is used at a time. Each microprogram contains the Alto emulator and I/O microcode, and additionally contains the emulator for one other language. That is, the "Mesa microcode" consists of the Alto emulator, the I/O microcode, and the microcode for emulating the Mesa language; similarly for the "Lisp microcode" and the "Smalltalk microcode". (There is also "Cedar microcode", which is presently the same as Mesa but will probably become different in the future.)

Microcode booting

When the Dorado is first turned on, it automatically goes through a complete bootstrap sequence that takes about a minute. (Most of this time is spent waiting for the disk to become ready.) If all goes well, at the end of this time the Mesa microcode is loaded and the Alto Operating System on disk partition 1 is started. This full bootstrap sequence can also be initiated by pressing the white button on the front panel of the Dorado (inside the cabinet, if the Dorado is cabinet-mounted).

Other forms of booting are initiated by pressing the boot button on the back of the keyboard, as detailed in the following paragraphs. There are single- and multiple-push boots. Due to software limitations, you must push the boot button for at least 0.25 second and no more than 2.5 seconds; multiple pushes must be spaced no more than 1.5 seconds apart; and the bootstrap operation does not commence until 1.5 seconds after you release the button for the last time.

1-push boot. This causes the currently-running microcode to go through the software bootstrap sequence, i.e., loading the Alto OS, or the NetExec if you have the BS and Quote keys pressed down, or whatever. A 1-push boot is the nearest equivalent to pressing the boot button on an Alto. It works only if the correct emulator microcode is already loaded and the Dorado is still running normally.

2-push boot. This causes the currently-running microcode to be forcibly restarted. It is similar to a 1-push boot, but it will work even if the Dorado has halted, and it does more extensive microcode initialization (in particular, it zeroes main memory, and it reverts to the default disk partition for the current emulator.) This does require, however, that there be intact emulator microcode in the Dorado's control store.

3-push boot. This initiates the complete microcode bootstrap sequence, similar to the automatic power-on boot. This is the normal way to switch from one emulator to another (see below). It assumes nothing about the current state of the machine.

4-push boot. This isn't really a boot: it causes the Dorado to shut down, a process that takes about 30 seconds. Shutting down the machine is necessary, for example, in order to change disk packs. When the machine is in the shut down state, pushing the boot button (one or more times) will initiate a complete microcode bootstrap, similar to a power-on or 3-push boot.

If you have no keys pressed down during a power-on boot or a 3-push boot, you always get the Mesa emulator. You may bootstrap other emulators by holding a key down during the boot:

C	for Cedar
L	for Lisp
M	for Mesa (same as all-keys-up boot)
S	for Smalltalk

For correct operation, you must hold the key down for at least 10 seconds after you initiate a 3-push boot.

These keys are independent of the ones used to select the desired *software* to run. The latter keys are the same as on the Alto; that is, to run the NetExec, you hold down the BS and Quote keys while booting. Thus, to bootstrap the Smalltalk microcode and then run the NetExec, you might have to hold down S, BS, and Quote and initiate a 3-push boot. Since this would require you to have at least three hands, some more convenient (though less mnemonic) alternate keys are defined for selecting the microcode to bootstrap:

=	for Cedar
[	for Lisp
RETURN	for Mesa (same as all-keys-up boot)
SHIFT	for Smalltalk

Microcode self-loading

The Dorado can also load its own microcode under program control, using a special instruction (LoadRam) described in section 2. There exists a program called LoadMB that runs on the Dorado and loads emulator microcode from a file on the Dorado's disk. The file must be in MB (micro binary) format, as produced by MicroD, and the microcode contained within that file must conform to certain conventions in order to be soft-bootable in this manner.

For example, the Mesa microcode is contained in a file called Mesa.MB. To load and start that microcode, type:

>LoadMB Mesa.MB

This loads the Dorado's control store with the contents of Mesa.MB and then starts it. That microcode (since it is a standard emulator) in turn loads and starts the Alto OS from disk partition 1, the default partition for Mesa.

This means of loading microcode isn't very interesting for bootstrapping the standard emulators, since the 3-push boot is easier and doesn't require the new microcode to be on your Dorado's disk. But it is useful for loading non-standard or experimental emulators. (The LoadMB program has some other features, described in section 3. The program is stored as [Ivy]<Dorado>LoadMB.run.)

Loading microcode using Midas

Each cluster of Dorados has an umbilical Alto that is able to control a connected Dorado for purposes such as hardware checkout and microcode debugging. The Alto program that performs these functions is called Midas.

Most Dorado users will have no occasion to use Midas. But if the normal microcode bootstrap mechanisms don't seem to be working, you can use Midas to load your desired emulator microcode.

If Midas is not already running, boot the Alto and type:

>Midas

Midas first displays a menu of the serial numbers of all the Dorados that are connected to this Alto. Select your Dorado by positioning the cursor over your Dorado's serial number and clicking RED (left or top mouse button). This step is skipped automatically if there is only one Dorado connected to the Alto.

If the selected Dorado is running, Midas displays a status of "Running" and puts up a menu (near the bottom of the screen) consisting of ABORT and DTACH items only. To halt the Dorado, select ABORT.

Now Midas displays a large menu near the bottom of the screen. Select RUN-PROGRAM, and the menu changes to a list of microprograms that you can run; these include CEDAR, LISP, MESA, and SMALLTALK. Select one of those. The screen then goes blank while Midas loads the Dorado's control store, a process that takes about a minute. After that, Midas starts the Dorado as if you had initiated a 2-push boot.

Important: *Do not leave Midas running on the Alto after you have loaded your microcode*, because 2-, 3-, and 4-push boots from the Dorado's keyboard don't work if Midas is controlling the Dorado. Therefore, exit Midas using SHIFT-SWAT (hold down the left SHIFT key and press the SWAT key, which is the blank key to the right of right SHIFT).

The versions of emulator microcode stored on the Midas Alto's disk might not always be up-to-date. If you are in doubt, update the microcode files by issuing the Executive command:

>@UpdateEmulators

which runs FTP to retrieve the current versions of all emulator microcode.

Machine status

There is a green light on the front panel of the Dorado that indicates the machine's present state, as follows:

1 flash. Bootstrap sequence in progress (power-on or 3-push boot).

2 flashes. Bootstrap sequence failed.

3 flashes. Transient power-supply problem—voltage or current was out-of-spec at least once since the last boot.

4 flashes. Present power-supply problem.

5 flashes. Dorado has been shut down by 4-push boot.

6 flashes. Dorado has overheated and shut itself down automatically.

7 flashes. Midas has taken over control of the Dorado.

Continuously on. Bootstrap sequence is believed to have completed successfully, and there have been no occurrences of problems indicated by 3 through 7 flashes.

Continuously off. Unknown but probably serious malfunction, or AC power turned off.

2. How it works

The Dorado contains a microcomputer that performs several functions, including monitoring power and temperature, turning on and off the main power supplies and the disk drive, and initiating bootstrap sequences. The microcomputer can sense the state of the boot button, and it controls the green light. This microcomputer and attendant EProms and other logic are mounted on the BaseBoard, the bottom logic board in the machine.

A 1-push boot is handled entirely by the Dorado microcode itself, with no action by the BaseBoard. Multiple-push and power-on boots are handled by the BaseBoard. A 3-push or power-on boot involves the following multiple-step sequence.

Loading Bootstrap microcode

The BaseBoard first halts the Dorado, issues an I/O reset, and resets or disables conditions that could interfere with initial bootstrapping (Hold, Fault task wakeup). It then forcibly loads into the Dorado's control store a small (~50 instruction) microprogram called Bootstrap. Finally, it starts the Dorado executing this microprogram.

Loading of Dorado microcode in this manner is very slow; therefore, the amount of microcode in Bootstrap is kept as small as possible.

Loading Initial microcode

Next, the Bootstrap microprogram and the BaseBoard microcomputer cooperate to load into the Dorado's control store a somewhat larger (~700 instruction) microprogram called Initial. Initial is stored as data in the BaseBoard's EProm. It is read by the microcomputer and handed to the Dorado (via CPReg) at relatively high speed; the Boostrap microprogram copies this data into the Dorado's control store.

When this operation is finished, the Boostrap microprogram gives control to the Initial microprogram it has loaded. Bootstrap and Initial are arranged to occupy disjoint regions of the control store. The BaseBoard then waits for the Dorado to complete the next step.

Initial execution

Initial performs a number of operations. First, it does a more thorough initialization of the Dorado. It corrects parity in all registers and memories. It initializes the memory system (cache, map, and storage) and puts it into a usable state. Finally, it notifies the BaseBoard that this step has been completed successfully. (This notification causes the BaseBoard to turn the green light continuously on. If this event fails to occur within a reasonable time, the BaseBoard switches to the 2-flash indication.)

Second, Initial enables the I/O tasks that it requires (Display, Ethernet, and Junk tasks). It then reads the keyboard in order to determine which emulator microcode to bootstrap.

Third, Initial makes contact with an Ethernet boot server (located in a Gateway or IFS system) and requests it to supply the selected emulator microcode. That microcode is transferred over the Ethernet and stored in the Dorado's main memory. (This step employs the Pup Microcode Boot protocol, which is also used for booting D0s and is documented in [Maxc]<Pup>EtherBoot.press.)

Finally, Initial calls the LoadRam procedure, which loads the emulator microcode from main memory into the Dorado's control store (IM and IFUM) and transfers control to its starting address. The emulator microcode replaces Bootstrap and Initial. LoadRam is a standard one-page microcode loading routine which is included in Initial and in every emulator. It occupies the same region of control store in each microprogram. Therefore, LoadRam can load a new microprogram without danger of overwriting itself.

Software bootstrap

When the emulator microcode is started—either as a result of the above sequence or by a 2-push boot—it does some further memory system initialization, including zeroing out all main storage. It resets the disk partition to the default one for that emulator. It initializes all I/O devices. Finally, it initiates an Alto-style software boot from either disk or Ethernet, according to the keys that are pressed down.

A 1-push boot is detected and handled by the emulator microcode itself, without action by the BaseBoard. It performs all actions described in the preceding paragraph except for zeroing memory and resetting the default disk partition.

The Alto-style software boot is described in Alto documentation: the Alto Hardware manual ([Maxc]<AltoDocs>AltoHardware.press, section 3.3), the BuildBoot documentation (<AltoDocs>BuildBoot.tty), and the Alto Boot protocol specification (<Pup>AltoBoot.press).

Microcode boot from disk

Eventually it will be possible to bootstrap emulator microcode from the disk as well as from the Ethernet, as is done for the D0. This will require some disk format changes and some other changes that are entangled with bringing up Pilot on the Dorado. Therefore, implementing microcode booting from the disk has been deferred.

3. How it is put together

This section attempts to locate all microcode and software required for Dorado bootstrapping and to set forth procedures that might otherwise be forgotten.

Bootstrap, Initial, and BaseBoard microcomputer code

The Bootstrap and Initial microprograms and the BaseBoard microcomputer software are programmed into a set of EProms that are installed on the BaseBoard. This programming is controlled by files on an Alto disk labelled "Sosinski—DoradoProms".

The sources and command files for the BaseBoard microcomputer code are kept in a dump-format file, [Ivy]<DoradoSource>DoradoBaseRom.dm. The Dorado microcode (Bootstrap.mb and Initial.mb) are kept in [Ivy]<Dorado>Bootstrap.dm. The following command files may be executed to perform the specified operations:

@DoradoBaseRom-Compile.cm@. This assembles all the BaseBoard microcomputer code, executes all of the following command files, and prints a pile of listings.

@GetNewDoradoUCode.cm@. This obtains Bootstrap.mb and Initial.mb from Ivy. It then runs a program called MBtoBase.run, which converts the Dorado microcode into a form that can be loaded along with the BaseBoard microcomputer code. (Bootstrap.mb is transformed into Boot0.mb and Initial.mb into Boot1.mb.)

@DoradoBaseRom-Load.cm@. This binds and loads all the microcomputer code, along with Boot0 and Boot1. Parameters in this command line control the placement of the various pieces (see below). The output is DoradoBaseRom.mb, which may be used as input both by APNew (for EProm programming) and Midas (for BaseBoard microcomputer code debugging).

@DoradoBaseRom-Blow.cm@. This controls the programming of the four BaseBoard EProms, which are Intel 2716s. The EProms have starting addresses and BaseBoard coordinates listed in the following table, and are programmed in the order given:

F000	c61
F800	b61
C000	f60
C800	e60

The microcomputer code and Bootstrap and Initial microprograms are placed independently so as to minimize the number of EProms that have to be reprogrammed when a change is made. Specifically, the microcomputer code and Bootstrap are contained in F000 and F800, while Initial and some tables that control loading of both Bootstrap and Initial are contained in C000 and C800. This has the following consequences:

If the microcomputer code is changed, F000 and F800 must be reprogrammed.

If Initial is changed, C000 and C800 must be reprogrammed.

If Bootstrap is changed (which is fairly unlikely), all four EProms must be reprogrammed.

The Bootstrap and Initial microcode are assembled from sources in [Ivy]<DoradoSource>BootstrapSources.dm. This must be done on a disk that already has the Alto emulator microcode environment, loaded from [Ivy]<Dorado>AEmu-dibs.dm.

LoadMB

The program LoadMB.run reads a Dorado MB-format file and does one of three things with it:

Immediately loads it into the Dorado's control store and starts it.

Creates an EB-format (Ether-bootable) file, for installation on boot servers.

Creates an SB-format (soft-bootable) file, for installation on the disk (whenever microcode booting from disk gets implemented).

Obviously, LoadMB can only run on a Dorado when performing the first function, but it can perform the latter two functions running on an Alto.

The complete form of the LoadMB command line is:

>LoadMB/*global-switches InputFile*.MB *parameter/switch* ...

If there are no global switches and no parameters, LoadMB simply loads the input file into the Dorado's control store and starts it. The global switches have the following effects:

/E	Writes the microcode onto an EB-format file, whose file name defaults to *InputFile*.EB.
/S	Writes the microcode onto an SB-format file, whose file name defaults to *InputFile*.SB.
/V	Pauses and awaits confirmation after reading the MB file and before loading the Dorado's control store.

The following parameters may be included on the command line:

filename/O	Specifies the output file name, overriding the default implied by the global switch. If neither /E nor /S has been specified, defaults to SB-format.
address/S	Specifies the octal starting address of the microcode. This defaults to 1076, the address of the label InitMap in all emulators.

For example, the command:

>LoadMB/E Mesa.MB DoradoMesa.EB/O 1076/S

reads microcode from Mesa.MB and creates an Ether-bootable file DoradoMesa.EB which has a starting address of 1076.

Installing emulator microcode on boot servers

At present, the only boot servers capable of booting Dorado and D0 microcode are the ones in Alto Gateways. (Adding this capability to the IFS boot servers is contemplated.) Installing microcode boot files is best done by a Gateway maintainer. The following instructions are for the Gateway maintainer's benefit and are therefore not tutorial.

Boot file numbers beginning with 3000 octal are set aside for microcode booting. The D0 uses numbers starting at 3000, and the Dorado starting at 3100. The current assignments for Dorado microcode boot files are:

3100	DoradoMesa.eb
3101	DoradoSmalltalk.eb
3102	DoradoLisp.eb
3103	DoradoCedar.eb

These correspondences must be established in the Gateway's GateParameter.txt file.

When a Dorado or D0 issues a microcode boot request, it specifies a boot file number in a space that begins with zero, and the server adds 3000 to it in order to select the correct boot file. Hence, the numbers 100, 101, 102, and 103 are wired into the Dorado's Initial microprogram for selecting Mesa, Smalltalk, Lisp, and Cedar respectively. However, the boot servers update these boot files

using the normal boot update protocol (as opposed to the special microcode boot protocol), and use the regular boot file numbers beginning at 3000 while doing so.

To update a microcode boot file, use LoadMB to create the desired EB-format file, and then use GateControl to store it on the nearest Alto Gateway.

LoadRam

LoadRam is a microcode routine that is present in Initial and in all emulators. It loads the Dorado's control store (IM and IFUM) with microcode taken from main memory, and optionally starts it. LoadRam may be invoked by opcodes in the Alto and Mesa instruction sets.

The microcode is taken from an indefinitely long array of Items in main memory. (An SB-format file consists of a version number followed by such an array. An EB-format file consists of an overhead page followed by the array.) Each Item is a 4-word block, as follows:

```
Item: TYPE = MACHINE DEPENDENT RECORD
  [
  extraIM: [0..17B], unused: [0..777B], type: [0..7B],
  addr: WORD,
  word0: WORD,
  word1: WORD
  ]

ItemType: TYPE = {IM, IFUM, End}
```

The array consists of any number of IM and IFUM items and is terminated by an End item.

In an IM item, addr is the absolute IM address, word0 and word1 are the left and right halves of the data, and extraIM contains the concatenation of LHParityBad, RStk[0], RHParityBad, and Block. In an IFUM item, addr is the IFUM address (including the instruction set number as the two high-order bits), and word0 and word1 are the left and right halves of the data.

In an End item, word0 is a checksum and word1 is the microcode starting address. The checksum should be such that the two's-complement sum of all the words of all the items (including the End item) is zero. LoadRam itself does not check the checksum; this is the responsibility of higher-level microcode or software. (For example, Initial checks the checksum of the microcode that it receives from a boot server.)

In the Mesa instruction set, LoadRam is opcode MISC 3, and it is invoked by:

```
LoadRam[itemArray: LONG POINTER TO Item, flag: BOOLEAN]
```

In the Alto instruction set, LoadRam is opcode 61036B, and it accepts a *short* pointer to the first Item in AC0 and the flag in AC1. (These instructions are defined identically on the D0, though of course the interpretation of the Items is different. As of this writing, the Alto LoadRam opcode has not yet been implemented on the D0.)

The interpretation of the flag is as follows. If flag is TRUE, LoadRam turns off tasking (and does in-line memory refresh on the D0), loads the control store from the Item array, and jumps to the starting address given in the End item with tasking still turned off. This operation is suitable for complete microcode replacement. The only requirement is that the new microcode have LoadRam in the same place, since LoadRam will refuse to overwrite itself.

If flag is FALSE, LoadRam leaves tasking on, loads the control store from the Item array, ignores the End block, and resumes normal emulation at the next opcode (Alto or Mesa). This operation is intended for microcode overlays. By convention, it must be assured that the new microcode is placed in such a way that it does not overwrite any existing microcode that is currently being executed by any task.